实验七 综合案例（5）

一、 实验目的

1、 掌握多态特性的定义与操作

二、 实验内容

1、PTA 作业《C++语言程序设计工程实践：面向过程高级》所有实验题

2、使用 Visual Studio 环境来编辑、编译和运行下列程序。

我们的程序存在两个不足,如下所述。

(1)各个账户对象无法通过数组来访问,使得需要分别对每个对象执行某个操作时，只能分别写出针对各对象的代码,无法用循环来完成。

(2）不同派生类的 deposit，withdraw，settle 等函数彼此独立,只有知道一个实例的具体类型后才能够调用这些函数。

本节中,将应用虚函数和抽象类对该程序进行改进，解决上述不足。本例对 Account 类做了如下改进。

(1)将 show 函数声明为虚函数,因此通过指向 CreditAccount 类实例的 Account 类型的指针来调用 show 函数时,被实际调用的将是为 CreditAccount 类定义的 show 函数。这样,如果创建-一个 Account 指针类型的数组,使各个元素分别指向各个账户对象,就可以通过一个循环来调用它们的 show 函数。

(2〉在 Account 类中添加 deposit, withdraw , settle 这 3 个函数的声明,且将它们都声明为纯虚函数,这使得通过基类的指针可以调用派生类的相应函数,而且无须给出它们在基类中的实现。经过这一改动之后，Account 类就变成了抽象类。

需要注意的是,虽然这几个函数在两个派生类中的原型相同,但两个派生类的 settle 函数的对外接口存在着隐式的差异。原因是储蓄账户一年结算一次,SavingsAccount 函数的 settle 函数接收的参数应该是每年 1 月 1 日,而信用账户一月结算一次,CreditAccount 函数的 settle 函数接收的参数应该是每月 1 日。而使用基类 Account 的指针来调用 settle 函数时,事先并不知道该指针所指向对象的具体类型,无法决定采用何种方式调用 settle 函数,因此只能将二者允许接收的参数统一为每月 1 日。同时对活期储蓄账户的 settle 函数进行修改,使它在结算利息之前先判断是否为 1 月,只有参数所给的日期是 1 月时才进行结算。

经过以上修改之后,由于对账户所做的各种操作都可以通过基类的指针来调用,因此可以把各账户对象的指针都放在一个数组中,只要给出数组索引就能够对几个账户对象进行操作。从本节开始,将综合实例的主函数改为由用户输入账户编号,对账户的操作类型和操作的参数,以增加程序的灵活性。

此外,我们使用运算符重载,对 Date 类进行了一个小修改-—将原先用来计算两日期相差天数的 distance 函数改为“一”运算符,使得计算两日期相差天数的写法更加直观,增加程序的可读性。

```
//date.h
#ifndef __DATE_H__
#define __DATE_H__

class Date {   //日期类
private:
    int year;       //年
    int month;          //月
    int day;        //日
    int totalDays;//该日期是从公元元年 1 月 1 日开始的第几天

public:
    Date(int year, int month, int day);    //用年、月、日构造日期
    int getYear() const { return year; }
    int getMonth() const { return month; }
    int getDay() const { return day; }
    int getMaxDay() const;              //获得当月有多少天
    bool isLeapYear() const {   //判断当年是否为闰年
        return year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
    }
    void show() const;                  //输出当前日期
    //计算两个日期之间差多少天
    int operator - (const Date& date) const {
        return totalDays - date.totalDays;
    }
};

#endif //__DATE_H__

//date.cpp
#include "date.h"
#include <iostream>
#include <cstdlib>
using namespace std;

namespace { //namespace 使下面的定义只在当前文件中有效
    //存储平年中某个月 1 日之前有多少天，为便于 getMaxDay 函数的实现，该数组多出一项
    const int DAYS_BEFORE_MONTH[] = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
}

Date::Date(int year, int month, int day) : year(year), month(month), day(day) {
    if (day <= 0 || day > getMaxDay()) {
```

```
            cout << "Invalid date: ";
            show();
            cout << endl;
            exit(1);
        }
        int years = year - 1;
        totalDays = years * 365 + years / 4 - years / 100 + years / 400
            + DAYS_BEFORE_MONTH[month - 1] + day;
        if (isLeapYear() && month > 2) totalDays++;
    }

    int Date::getMaxDay() const {
        if (isLeapYear() && month == 2)
            return 29;
        else
            return                                  DAYS_BEFORE_MONTH[month]-
DAYS_BEFORE_MONTH[month - 1];
    }

    void Date::show() const {
        cout << getYear() << "-" << getMonth() << "-" << getDay();
}

    //accumulator.h
    #ifndef __ACCUMULATOR_H__
    #define __ACCUMULATOR_H__
    #include "date.h"

    class Accumulator {    //将某个数值按日累加
    private:
        Date lastDate;     //上次变更数值的时期
        double value;      //数值的当前值
        double sum;        //数值按日累加之和
    public:
        //构造函数，date 为开始累加的日期，value 为初始值
        Accumulator(const Date &date, double value)
            : lastDate(date), value(value), sum(0) { }

        //获得到日期 date 的累加结果
        double getSum(const Date &date) const {
            return sum + value * (date - lastDate);
        }

        //在 date 将数值变更为 value
```

```
        void change(const Date &date, double value) {
            sum = getSum(date);
            lastDate = date;
            this->value = value;
        }

        //初始化，将日期变为 date，数值变为 value，累加器清零
        void reset(const Date &date, double value) {
            lastDate = date;
            this->value = value;
            sum = 0;
        }
    };

#endif //__ACCUMULATOR_H__
```

```
    //account.h
    #ifndef __ACCOUNT_H__
    #define __ACCOUNT_H__
    #include "date.h"
    #include "accumulator.h"
    #include <string>

    class Account { //账户类
    private:
        std::string id; //帐号
        double balance;  //余额
        static double total; //所有账户的总金额
    protected:
        //供派生类调用的构造函数，id 为账户
        Account(const Date &date, const std::string &id);
        //记录一笔帐，date 为日期，amount 为金额，desc 为说明
        void record(const Date &date, double amount, const std::string &desc);
        //报告错误信息
        void error(const std::string &msg) const;
    public:
        const std::string &getId() const { return id; }
        double getBalance() const { return balance; }
        static double getTotal() { return total; }
        //存入现金，date 为日期，amount 为金额，desc 为款项说明
        virtual void deposit(const Date &date, double amount, const std::string
&desc) = 0;
        //取出现金，date 为日期，amount 为金额，desc 为款项说明
```

```
        virtual void withdraw(const Date &date, double amount, const std::string &desc) = 0;
        //结算（计算利息、年费等），每月结算一次，date 为结算日期
        virtual void settle(const Date &date) = 0;
        //显示账户信息
        virtual void show() const;
    };

    class SavingsAccount : public Account { //储蓄账户类
    private:
        Accumulator acc; //辅助计算利息的累加器
        double rate;        //存款的年利率
    public:
        //构造函数
        SavingsAccount(const Date &date, const std::string &id, double rate);
        double getRate() const { return rate; }
        virtual void deposit(const Date &date, double amount, const std::string &desc);
        virtual void withdraw(const Date &date, double amount, const std::string &desc);
        virtual void settle(const Date &date);
    };

    class CreditAccount : public Account { //信用账户类
    private:
        Accumulator acc; //辅助计算利息的累加器
        double credit;          //信用额度
        double rate;        //欠款的日利率
        double fee;             //信用卡年费

        double getDebt() const {   //获得欠款额
            double balance = getBalance();
            return (balance < 0 ? balance : 0);
        }
    public:
        //构造函数
        CreditAccount(const Date &date, const std::string &id, double credit, double rate, double fee);
        double getCredit() const { return credit; }
        double getRate() const { return rate; }
        double getFee() const { return fee; }
        double getAvailableCredit() const { //获得可用信用
            if (getBalance() < 0)
                return credit + getBalance();
```

```
            else
                return credit;
        }
        virtual void deposit(const Date &date, double amount, const std::string
&desc);
        virtual void withdraw(const Date &date, double amount, const std::string
&desc);
        virtual void settle(const Date &date);
        virtual void show() const;
    };

#endif //__ACCOUNT_H__


    //account.cpp
    #include "account.h"
    #include <cmath>
    #include <iostream>
    using namespace std;

    double Account::total = 0;

    //Account 类的实现
    Account::Account(const Date &date, const string &id)
        : id(id), balance(0) {
        date.show();
        cout << "\t#" << id << " created" << endl;
    }

    void Account::record(const Date &date, double amount, const string &desc) {
        amount = floor(amount * 100 + 0.5) / 100;   //保留小数点后两位
        balance += amount;
        total += amount;
        date.show();
        cout << "\t#" << id << "\t" << amount << "\t" << balance << "\t" << desc
<< endl;
    }

    void Account::show() const {
        cout << id << "\tBalance: " << balance;
    }

    void Account::error(const string &msg) const {
        cout << "Error(#" << id << "): " << msg << endl;
```

```
}

//SavingsAccount 类相关成员函数的实现
SavingsAccount::SavingsAccount(const Date &date, const string &id, double rate)
    : Account(date, id), rate(rate), acc(date, 0) { }

void SavingsAccount::deposit(const Date &date, double amount, const string &desc) {
    record(date, amount, desc);
    acc.change(date, getBalance());
}

void SavingsAccount::withdraw(const Date &date, double amount, const string &desc) {
    if (amount > getBalance()) {
        error("not enough money");
    } else {
        record(date, -amount, desc);
        acc.change(date, getBalance());
    }
}

void SavingsAccount::settle(const Date &date) {
    if (date.getMonth() == 1) {//每年的一月计算一次利息
        double interest = acc.getSum(date) * rate
            / (date - Date(date.getYear() - 1, 1, 1));
        if (interest != 0)
            record(date, interest, "interest");
        acc.reset(date, getBalance());
    }
}

//CreditAccount 类相关成员函数的实现
CreditAccount::CreditAccount(const Date& date, const string& id, double credit, double rate, double fee)
    : Account(date, id), credit(credit), rate(rate), fee(fee), acc(date, 0) { }

void CreditAccount::deposit(const Date &date, double amount, const string &desc) {
    record(date, amount, desc);
    acc.change(date, getDebt());
}
```

```
void CreditAccount::withdraw(const Date &date, double amount, const string &desc) {
    if (amount - getBalance() > credit) {
        error("not enough credit");
    } else {
        record(date, -amount, desc);
        acc.change(date, getDebt());
    }
}

void CreditAccount::settle(const Date &date) {
    double interest = acc.getSum(date) * rate;
    if (interest != 0)
        record(date, interest, "interest");
    if (date.getMonth() == 1)
        record(date, -fee, "annual fee");
    acc.reset(date, getDebt());
}

void CreditAccount::show() const {
    Account::show();
    cout << "\tAvailable credit:" << getAvailableCredit();
}

//main.cpp
#include "account.h"
#include <iostream>
using namespace std;

int main() {
    Date date(2008, 11, 1);      //起始日期
    //建立几个账户
    SavingsAccount sa1(date, "S3755217", 0.015);
    SavingsAccount sa2(date, "02342342", 0.015);
    CreditAccount ca(date, "C5392394", 10000, 0.0005, 50);
    Account *accounts[] = { &sa1, &sa2, &ca };
    const int n = sizeof(accounts) / sizeof(Account*); //账户总数

    cout << "(d)deposit (w)withdraw (s)show (c)change day (n)next month (e)exit" << endl;
    char cmd;
    do {
        //显示日期和总金额
        date.show();
```

```
        cout << "\tTotal: " << Account::getTotal() << "\tcommand> ";

        int index, day;
        double amount;
        string desc;

        cin >> cmd;
        switch (cmd) {
        case 'd': //存入现金
            cin >> index >> amount;
            getline(cin, desc);
            accounts[index]->deposit(date, amount, desc);
            break;
        case 'w': //取出现金
            cin >> index >> amount;
            getline(cin, desc);
            accounts[index]->withdraw(date, amount, desc);
            break;
        case 's':  //查询各账户信息
            for (int i = 0; i < n; i++) {
                cout << "[" << i << "] ";
                accounts[i]->show();
                cout << endl;
            }
            break;
        case 'c':  //改变日期
            cin >> day;
            if (day < date.getDay())
                cout << "You cannot specify a previous day";
            else if (day > date.getMaxDay())
                cout << "Invalid day";
            else
                date = Date(date.getYear(), date.getMonth(), day);
            break;
        case 'n': //进入下个月
            if (date.getMonth() == 12)
                date = Date(date.getYear() + 1, 1, 1);
            else
                date = Date(date.getYear(), date.getMonth() + 1, 1);
            for (int i = 0; i < n; i++)
                accounts[i]->settle(date);
            break;
        }
    } while (cmd != 'e');
```

```
    return 0;
}
```

三、 实验要求

1、 完成 PTA 作业《C++语言程序设计工程实践：面向对象高级》的所有实验题。

2、 成功在 Visual Studio 2019 里面运行上述代码。